Intuitive Performance Engineering at the Exascale with TAU and TAU Commander

John C. Linford
ParaTools, Inc.

Argonne Extreme Scale Computing Training Program
12 August 2015, Pheasant Run “Resort”
ParaTools Accelerates Software

National Security
• Cyber security
• Weapons research
• Intelligence
• Contingency

Research
• Simulation
• Analysis
• Testing

Engineering & Manufacturing
• Time-to-market
• Quality

Policy
• Energy
• Environment
• Emergency

Copyright © ParaTools, Inc.
Value Proposition

First in Discovery

Lower Operating Costs

Improve Capability

Lower Time-to-Solution

“Performance Engineering”
Overview

- Overview
- TAU Overview
- Case Studies
- TAU Commander
Intuitive Performance Engineering

OVERVIEW
The Metrics We Care About

Performance

Efficiency

Productivity
The TAU Performance System®

• **Integrated toolkit** for performance problem solving
  – Instrumentation, measurement, analysis, visualization
  – Portable profiling and tracing
  – Performance data management and data mining
• Direct and indirect measurement
• **Free, open source, BSD license**
• Available on all HPC platforms (and some non-HPC)
• [http://tau.uoregon.edu/](http://tau.uoregon.edu/)
How do we Improve Productivity?
TAU Commander

An intuitive interface to
the TAU Performance System
Intuitive Performance Engineering

THE TAU PERFORMANCE SYSTEM
• Tuning and Analysis Utilities (20+ year project)

• Comprehensive performance profiling and tracing
  – Integrated, scalable, flexible, portable
  – Targets all parallel programming-execution paradigms

• Integrated performance toolkit
  – Instrumentation, measurement, analysis, visualization
  – Widely-ported performance profiling / tracing system
  – Performance data management and data mining
  – Open source (BSD-style license)

• Integrates with application frameworks
TAU Supports All HPC Platforms

- C/C++
- Fortran
- pthreads
- Intel
- GNU
- MinGW
- CUDA
- UPC
- OpenACC
- Intel MIC
- LLVM
- PGI
- GPU
- Java
- MPI
- OpenMP
- Cray
- Sun
- AIX
- ARM
- Linux
- Windows
- Fujitsu
- Android
- MPC
- OS X

Insert yours here
Measurement Approaches

Profiling

Shows how much time was spent in each routine

Tracing

Shows when events take place on a timeline
Types of Performance Profiles

- **Flat** profiles
  - Metric (e.g., time) spent in an event
  - Exclusive/inclusive, # of calls, child calls, ...

- **Callpath** profiles
  - Time spent along a calling path (edges in callgraph)
  - “main=> f1 => f2 => MPI_Send”
  - Set the **TAU_CALLPATH_DEPTH** environment variable

- **Phase** profiles
  - Flat profiles under a phase (nested phases allowed)
  - Default “main” phase
  - Supports static or dynamic (e.g. per-iteration) phases
How much data do you want?

Limited Profile  Loop Profile  Callpath Profile

O(KB)  Flat Profile  Phase Profile  Trace  O(TB)

All levels support multiple metrics/counters
Performance Data Measurement

Direct via Probes

- Exact measurement
- Fine-grain control
- Calls inserted into code

```
call TAU_START('potential')
// code
call TAU_STOP('potential')
```

Indirect via Sampling

- No code modification
- Minimal effort
- Relies on debug symbols (-g option)
• Use TAU’s compiler wrappers
  • Replace CXX with tau_cxx.sh, etc.
  • Automatically instruments source code, links with TAU libraries.
• Use tau_cc.sh for C, tau_f90.sh for Fortran, etc.

Makefile without TAU

```makefile
CXX = mpicxx
F90 = mpif90
CXXFLAGS =
LIBS = -lm
OBJJS = f1.o f2.o f3.o ... fn.o

app: $(OBJJS)
    $(CXX) $(LDFLAGS) $(OBJJS) -o $@
    $(LIBS)
.cpp.o:
    $(CXX) $(CXXFLAGS) -c $<
```

Makefile with TAU

```makefile
CXX = tau_cxx.sh
F90 = tau_f90.sh
CXXFLAGS =
LIBS = -lm
OBJJS = f1.o f2.o f3.o ... fn.o

app: $(OBJJS)
    $(CXX) $(LDFLAGS) $(OBJJS) -o $@
    $(LIBS)
.cpp.o:
    $(CXX) $(CXXFLAGS) -c $<
```
Performance Engineering Workflow

**Instrumentation**
- **Source**
  - C, C++, Fortran, UPC, ...
  - Python, Java, ...
  - Robust parsers (PDT)
- **Library**
  - Interposition (MPI, GASNET, …)
  - Wrapper generation
- **Linker**
  - Static, Dynamic
  - Preloading (LD_PRELOAD)
- **Executable**
  - Dynamic (Dyninst)
  - Binary (Dinist, MAQAO, PEBIL)

**Measurement**
- **Events**
  - Static, Dynamic
  - Routine, Block, Loop
  - Threading, Communication
  - Heterogeneous
- **Profiling**
  - Flat, Callpath, Phase, Snapshot
  - Probe, Sampling, Compiler, Hybrid
- **Tracing**
  - TAU, Scalasca, ScoreP
  - Open Trace Format (OTF)
- **Metadata**
  - System
  - User defined

**Analysis**
- **Profiles**
  - ParaProf analyzer & visualizer
    - 3D profile data visualization
    - Communication matrix
    - Callstack analysis
    - Graph generation
    - PerfDMF
    - PerfExplorer profile data miner
- **Traces**
  - OTF, SLOG-2
  - Vampir
  - Jumpshot
- **Online**
  - Event unification
  - Statistics calculation
Instrument: Add Probes

- **Source code** instrumentation
  - PDT parsers, pre-processors

- **Wrap** external libraries
  - I/O, MPI, Memory, CUDA, OpenCL, pthread

- **Rewrite** the binary executable
  - Dyninst, MAQAO
Measure: Gather Data

• Direct measurement via *probes*

• Indirect measurement via *sampling*

• Throttling and runtime control

• Interface with external packages (PAPI)
Analyze: Synthesize Knowledge

- Data *visualization*

- Data *mining*

- Statistical analysis

- Import/export performance data
Using TAU: A Brief Introduction

- Each configuration of TAU corresponds to a unique stub makefile (*TAU_MAKEFILE*) in the TAU installation directory

```bash
% ls /soft/perftools/tau/tau_latest/bgq/lib/Makefile.*
Makefile.tau-bgqtimers-mpi-pdt-openmp-opari
Makefile.tau-bgqtimers-mpi-pthread-pdt
Makefile.tau-bgqtimers-papi-mpi-pdt
Makefile.tau-bgqtimers-papi-mpi-pdt-openmp-opari
Makefile.tau-bgqtimers-papi-mpi-pthread-pdt
Makefile.tau-bgqtimers-pdt
Makefile.tau-papi-mpi-pdt-openmp-opari
Makefile.tau-papi-mpi-pdt-openmp-opari-scorep
Makefile.tau-papi-mpi-pdt-scorep
```
1. Choose an appropriate TAU_MAKEFILE:

```
% soft add +tau-latest
% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/
  bgq/lib/Makefile.tau-bgqtimers-mpi-pdt
% export TAU_OPTIONS=’-optVerbose …’
  #(see tau_compiler.sh -help for more options)
```

2. Use tau_f90.sh, tau_cxx.sh, etc. as Fortran, C++, etc. compiler:

```
% mpixlf90_r foo.f90
changes to
% tau_f90.sh foo.f90
```

3. Execute application:

```
% qsub –A <queue> –q R.bc –n 256 –t 10 ./a.out
Note: If TAU_MAKEFILE has “papi” in its name, set TAU_METRICS:
% qsub --env TAU_METRICS=BGQ_TIMERS:PAPI_L2_DCM...
```

4. Analyze performance data:

```
pprof       (for text based profile display)
paraprof    (for GUI)
```
% ssh mira.alcf.anl.gov
% tar xzvf /soft/perftools/tau/workshop.tgz
% cd workshop
% less README

For an MPI+F90 application, you may want to start with:
% soft add +tau-latest
% export TAU_MAKEFILE=
   $TAU/Makefile.tau-bgqtimers-papi-mpi-pdt
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
% qsub -q R.bc -n 2 --mode c16 -t 10 -A ... ./a.out
% paraprof
How Much Time per Code Region?

% paraprof (Click on label, e.g. “Mean” or “node 0”)
How Many Instructions per Code Region?

%!paraprof (Options → Select Metric... → Exclusive... → PAPI_FP_INS)
How Many L1 or L2 Cache Misses?

% paraprof (Options → Select Metric... → Exclusive... → PAPI_L1_DCM)
# How Much Memory Does the Code Use?

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.TAU application</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>14,236,992.16</td>
<td>27,169.781</td>
<td>49,152</td>
<td>1</td>
<td>524.001</td>
<td>2,013.103</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>13,132,932</td>
<td>23,292</td>
<td>262,144</td>
<td>1</td>
<td>563.839</td>
<td>4,492.057</td>
</tr>
<tr>
<td><strong>MPI_Finalize()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OurMain()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>1,298,918.679</td>
<td>1,495.125</td>
<td>461,766.25</td>
<td>4</td>
<td>868.769</td>
<td>16,928.073</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>48,150</td>
<td>20</td>
<td>36,032</td>
<td>11</td>
<td>2,407.5</td>
<td>7,911.992</td>
</tr>
<tr>
<td><strong>&lt;module&gt;</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>3,465</td>
<td>9</td>
<td>769</td>
<td>32</td>
<td>385</td>
<td>260.2</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>4,314</td>
<td>12</td>
<td>769</td>
<td>32</td>
<td>359.5</td>
<td>240.981</td>
</tr>
<tr>
<td><strong>&lt;module&gt;</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Utilization (heap, in KB)</td>
<td>849,270.344</td>
<td>192,825.168</td>
<td>0.078</td>
<td>147,832.141</td>
<td>62,621.576</td>
<td></td>
</tr>
<tr>
<td>Message size for all-gather</td>
<td>4,096</td>
<td>1</td>
<td>4,096</td>
<td>4,096</td>
<td>4,096</td>
<td>0</td>
</tr>
<tr>
<td>Message size for all-reduce</td>
<td>23,340</td>
<td>843</td>
<td>320</td>
<td>4</td>
<td>27.687</td>
<td>64.653</td>
</tr>
<tr>
<td>Message size for all-to-all</td>
<td>104</td>
<td>26</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Message size for broadcast</td>
<td>24,923</td>
<td>206</td>
<td>8,788</td>
<td>4</td>
<td>120.985</td>
<td>860.992</td>
</tr>
<tr>
<td>Message size for reduce</td>
<td>8,912</td>
<td>8</td>
<td>8,788</td>
<td>4</td>
<td>1,114</td>
<td>2,900.511</td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>27,417,881,391.51</td>
<td>413,600.719</td>
<td>24,025,667</td>
<td>1</td>
<td>66,290.701</td>
<td>199,538.234</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>27,468,709,355.914</td>
<td>435,669.625</td>
<td>24,025,667</td>
<td>0</td>
<td>63,049.402</td>
<td>195,561.193</td>
</tr>
</tbody>
</table>

% paraprof (Right-click label [e.g “node 0”] → Show Context Event Window)
How Much Memory Does the Code Use?

![TAU: ParaProf: Mean Context Events - sphere_np32_nsteps5_mem.pck](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>free size (bytes)</td>
<td>14,236,992.16</td>
<td>27,169,781</td>
<td>49,152</td>
<td>1</td>
<td>524.001</td>
<td>2,013.103</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>13,132,932</td>
<td>23,292</td>
<td>262,144</td>
<td>1</td>
<td>563.839</td>
<td>4,492.057</td>
</tr>
<tr>
<td><strong>MPI_Finalize</strong>()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>1,298,918.679</td>
<td>1,495.125</td>
<td>461,766.25</td>
<td>4</td>
<td>868.769</td>
<td>16,928.073</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>48,150</td>
<td>20</td>
<td>36,032</td>
<td>11</td>
<td>2,407.5</td>
<td>7,911.992</td>
</tr>
<tr>
<td><strong>OurMain</strong>()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>3,465</td>
<td>9</td>
<td>769</td>
<td>32</td>
<td>385</td>
<td>260.2</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>4,314</td>
<td>12</td>
<td>769</td>
<td>32</td>
<td>359.5</td>
<td>240.981</td>
</tr>
<tr>
<td><strong>&lt;module&gt;</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>293,088</td>
<td>449</td>
<td>32,564</td>
<td>32</td>
<td>652.757</td>
<td>1,526.875</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>311,966</td>
<td>493</td>
<td>32,564</td>
<td>32</td>
<td>632.791</td>
<td>1,460.941</td>
</tr>
<tr>
<td>staticCFD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>init</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;module&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Utilization (heap, in KB)</td>
<td>849,270.344</td>
<td>192,825.168</td>
<td>0.078</td>
<td>147,832.141</td>
<td>62,621.576</td>
<td></td>
</tr>
<tr>
<td>Message size for all-gather</td>
<td>4,096</td>
<td>1</td>
<td>4,096</td>
<td>4,096</td>
<td>4,096</td>
<td>0</td>
</tr>
<tr>
<td>Message size for all-reduce</td>
<td>23,340</td>
<td>843</td>
<td>320</td>
<td>4</td>
<td>27.687</td>
<td>64.653</td>
</tr>
<tr>
<td>Message size for all-to-all</td>
<td>4104</td>
<td>26</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Message size for broadcast</td>
<td>24,923</td>
<td>206</td>
<td>8,788</td>
<td>4</td>
<td>120.985</td>
<td>860.992</td>
</tr>
<tr>
<td>Message size for reduce</td>
<td>4,912</td>
<td>8</td>
<td>8,788</td>
<td>4</td>
<td>1,114</td>
<td>2,900.511</td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>27,417,881,391.51</td>
<td>413,600.719</td>
<td>24,025,667</td>
<td>1</td>
<td>66,290.701</td>
<td>199,538.234</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>27,468,709,355.914</td>
<td>435,669.625</td>
<td>24,025,667</td>
<td>0</td>
<td>63,049.402</td>
<td>195,561.193</td>
</tr>
</tbody>
</table>

% **paraprof**  (Right-click label [e.g “node 0”] → Show Context Event Window)
Where is Memory Allocated / Decompiled?

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU application</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>14,236,992.16</td>
<td>27,169.781</td>
<td>49,152</td>
<td>1</td>
<td>524.001</td>
<td>2,013.103</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>13,132,932</td>
<td>23,292</td>
<td>262,144</td>
<td>1</td>
<td>563.839</td>
<td>4,492.057</td>
</tr>
<tr>
<td>MPI_Finalize()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OurMain()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>1,298,918.679</td>
<td>1,495.125</td>
<td>461,766.25</td>
<td>4</td>
<td>868.769</td>
<td>16,928.073</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>48,150</td>
<td>20</td>
<td>36,032</td>
<td>11</td>
<td>2,407.5</td>
<td>7,911.992</td>
</tr>
<tr>
<td>&lt;module&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>3,465</td>
<td>9</td>
<td>769</td>
<td>32</td>
<td>385</td>
<td>260.2</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>4,314</td>
<td>12</td>
<td>769</td>
<td>32</td>
<td>359.5</td>
<td>240.981</td>
</tr>
<tr>
<td>staticCFD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>init</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;module&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Utilization (heap, in KB)</td>
<td>849,270.344</td>
<td>192,825.168</td>
<td>0.078</td>
<td>147,832.141</td>
<td>62,621.576</td>
<td></td>
</tr>
<tr>
<td>Message size for all-gather</td>
<td>4,096</td>
<td>1</td>
<td>4,096</td>
<td>4,096</td>
<td>4,096</td>
<td>0</td>
</tr>
<tr>
<td>Message size for all-reduce</td>
<td>23,340</td>
<td>843</td>
<td>320</td>
<td>4</td>
<td>27.687</td>
<td>64.653</td>
</tr>
<tr>
<td>Message size for all-to-all</td>
<td>104</td>
<td>26</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Message size for broadcast</td>
<td>24,923</td>
<td>206</td>
<td>8,788</td>
<td>4</td>
<td>120.985</td>
<td>860.992</td>
</tr>
<tr>
<td>Message size for reduce</td>
<td>8,912</td>
<td>8</td>
<td>8,788</td>
<td>4</td>
<td>1,114</td>
<td>2,900.511</td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>27,417,881,391.51</td>
<td>413,600.719</td>
<td>24,025,667</td>
<td>1</td>
<td>66,290.701</td>
<td>199,538.234</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>27,468,709,355.914</td>
<td>435,669.625</td>
<td>24,025,667</td>
<td>0</td>
<td>63,049.402</td>
<td>195,561.193</td>
</tr>
</tbody>
</table>

%% paraprof (Right-click label [e.g “node 0”] → Show Context Event Window)
What are the I/O Characteristics?

Write bandwidth per file

Bytes written to each file

% paraprof (Right-click label [e.g “node 0”] → Show Context Event Window)
What are the I/O Characteristics?

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initialize</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LoadBodyEuler</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LoadMesh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI-IO Bytes Written</td>
<td>4,328,712</td>
<td>144</td>
<td>893,152</td>
<td>0</td>
<td>30,060.5</td>
<td>128,042.696</td>
</tr>
<tr>
<td>MPI-IO Write Bandwidth (MB/s)</td>
<td>144</td>
<td>196.86</td>
<td>0</td>
<td>3.421</td>
<td>16.87</td>
<td></td>
</tr>
<tr>
<td>MPI_Allgatherv()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Bcast()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Comm_create()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_File_close()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_File_open()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_File_write_all()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_File_write_at()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Finalize()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Gather()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Gatherv()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

% **paraprof** (Right-click label [e.g “node 0”] → Show Context Event Window)
How Much Time is spent in Collectives?

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>NumMax</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Wait()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message size for all-gather</td>
<td>305,753,268</td>
<td>72</td>
<td>172,215,296</td>
<td>4</td>
<td>4,246,573.167</td>
<td>22,551,605.859</td>
</tr>
<tr>
<td>Message size for all-reduce</td>
<td>163,308</td>
<td>632</td>
<td>21,908</td>
<td>4</td>
<td>258.399</td>
<td>897.725</td>
</tr>
<tr>
<td>Message size for all-to-all</td>
<td>112</td>
<td>14</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Message size for broadcast</td>
<td>692,208,045.5</td>
<td>3,346</td>
<td>18,117,620</td>
<td>0</td>
<td>206,876.284</td>
<td>1,284,673.036</td>
</tr>
<tr>
<td>Message size for gather</td>
<td>6,901,452.378</td>
<td>15.312</td>
<td>1,387,306.625</td>
<td>4</td>
<td>450,707.094</td>
<td>483,216.499</td>
</tr>
<tr>
<td>Message size for reduce</td>
<td>66,812</td>
<td>1,520</td>
<td>56</td>
<td>4</td>
<td>43.955</td>
<td>21.598</td>
</tr>
<tr>
<td>Message size for scatter</td>
<td>63,147.906</td>
<td>146</td>
<td>62,567.906</td>
<td>4</td>
<td>432.52</td>
<td>5,160.063</td>
</tr>
</tbody>
</table>

Metric: TIME
Value: Exclusive
Units: seconds

- MPI_Bcast(): 37.532
- MPI_Allreduce(): 16.969
- MPI_Reduce(): 3.364
- MPI_Scatter(): 1.649
- MPI_Gather(): 1.622
- MPI_Gatherv(): 0.984
- MPI_Allgather(): 0.758
- MPI_Allgatherv(): 0.636
- MPI_Alltoall(): 0.041
- MPI_Scatterv(): 0.006
- MPI_Barrier(): 0.004
3D Profile Visualization

% paraprof (Windows → 3D Visualization)
% qsub –env TAU_COMM_MATRIX=1 ...
% paraprof (Windows → 3D Communication Matrix)
3D Topology Visualization

% paraprof (Windows → 3D Visualization → Topology Plot)
How Does Each Routine Scale?

% perfexplorer (Charts → Runtime Breakdown)
How Does Each Routine Scale?

% perfexplorer (Charts → Stacked Bar Chart)
Which Events Correlate with Runtime?

% perfexplorer (Charts ➔ Correlate Events with Total Runtime)
When do Events Occur?
To generate a trace and visualize it in Jumpshot:

```bash
% qsub -env TAU_TRACE=1 ...
% tau_treemerge.pl
% tau2slog2 tau.trc tau.edf -o app.slog2
% jumpshot app.slog2
```
What Caused My Application to Crash?

% qsub -env TAU_TRACK_SIGNALS=1 ...
% paraprof
What Caused My Application to Crash?

Right-click to see source code
What Caused My Application to Crash?

Error shown in ParaProf Source Browser
Intuitive Performance Engineering

CASE STUDIES
Strand Technology

Technology Drivers

• **Timeliness** (automation of mesh generation)
• **Timeliness** (automation and scalability of domain connectivity)
• **Timeliness/Physical accuracy** (computational efficiency and scalability of aerodynamic solvers)
• **Processor architecture** (small memory footprint maps well to hierarchical memory architectures, e.g., multi-core, GPU)

CREATE-AV Example

This is a new meshing paradigm introduced in 2007 by current members of the CREATE-AV technical staff. The technology is being matured in the Helios product and will be deployed through both Helios and Kestrel.
Target Platforms

Armstrong [XC30]

Haise [iDataPlex]

Kilrain [iDataPlex]

Lightning [XC30]

Cori
Initial Profile on Babbage

MPI_Barrier

MPI_Send

File I/O

Useful Work!
Hot Spot Optimization

Useful work!

MPI_Waitall

Useful work!

File I/O
65% Runtime Reduction (~2x faster)
Cray XC30

Slower! What happened???
No worries, I fix it
• INCITE magnetohydrodynamics simulation to understand solar winds and coronal heating
  – First direct numerical simulations of Alfvén wave (AW) turbulence in extended solar atmosphere accounting for inhomogeneities
  – Team
    • University of New Hampshire (Jean Perez and Benjamin Chandran)
    • ALCF (Tim Williams)
    • University of Oregon (Sameer Shende)
• IRMHD (Inhomogeneous Reduced Magnetohydrodynamics)
  – Fortran 90 and MPI
  – Excellent weak and strong scaling properties
  – Tested and benchmarked on Intrepid and Mira
• HPC Source article and ALCF news
IRMHD Communication Analysis

• Source-based (direct) instrumentation
• MPI instrumentation and volume measurement
• IRMHD exhibited significant synchronous communication bottlenecks
• On 2,408 cores of BG/P:
  – MPI_Send and MPI_Bcast take significant time
  – Opportunities for communication/computation overlap
  – Identified possible targets for computation improvements
• On 2,408 cores, overall execution time reduced from 528.18 core hours to 70.8 core hours (>7x improvement)
• Non-blocking communication substrate
• More efficient implementation of underlying FFT
IRMHD Optimization on MIRA (BG/Q)

- Oversubscribe nodes: 32k ranks vs. 16k per node
- Overall time improvement: 71.23% of original
Intuitive Performance Engineering

TAU COMMANDER
TAU: Powerful and Complex

How do we navigate?
The TAU Commander Approach

• Say where you’re going, not how to get there
• **TAU Projects** give context to the user’s actions
  – Defines desired metrics and measurement approach
  – Defines operating environment
  – Establishes a baseline for error checking
T-A-M Model for Performance Engineering

- **Target**
  - Installed software
  - Available compilers
  - Host architecture/OS
- **Application**
  - MPI
  - CUDA
- **Measurement**
  - Profile, trace, or both
  - Sample, source inst.

TAU Experiment = (Target, Application, Measurement)
TAU Commander GUI

Dashboard

Last Project

Quick Start

I want to:
Profile

an application named:
Application Name

on:
Target...
New target name...

Make it so

Recent Projects

Last Modified: Tuesday
Data Size: 100MB
TAU Commander CLI

This command’s usage

Subcommand usage

Shortcuts

This command’s usage

Subcommand usage

Shortcuts

This command’s usage

Subcommand usage

Shortcuts
## TAU Commander CLI Dashboard

```bash
jlinford@east03 ~/workspace/tauclmr/examples/mm $ tau dash
```

### Targets (/home/jlinford/.tau)

<table>
<thead>
<tr>
<th>Name</th>
<th>Host OS</th>
<th>Host Arch</th>
<th>C</th>
<th>Fortran</th>
<th>In Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost</td>
<td>Linux</td>
<td>x86_64</td>
<td>/usr/bin/gcc</td>
<td>/usr/bin/g++</td>
<td>/usr/bin/gfortran</td>
</tr>
</tbody>
</table>

### Applications (/home/jlinford/.tau)

<table>
<thead>
<tr>
<th>Name</th>
<th>OpenMP</th>
<th>Pthreads</th>
<th>MPI</th>
<th>CUDA</th>
<th>MIC</th>
<th>SHMEM</th>
<th>MPC</th>
<th>In Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex-mm-serial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ex-mm</td>
</tr>
<tr>
<td>ex-mm-openmp</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ex-mm</td>
</tr>
<tr>
<td>ex-mm-openmp-mpi</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ex-mm</td>
</tr>
</tbody>
</table>

### Measurements (/home/jlinford/.tau)

<table>
<thead>
<tr>
<th>Name</th>
<th>Profile</th>
<th>Trace</th>
<th>Sample</th>
<th>Source Inst.</th>
<th>Compiler Inst.</th>
<th>MPI</th>
<th>OpenMP</th>
<th>Callpath</th>
<th>Mem. Usage</th>
<th>Mem. Alloc</th>
<th>In Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>exPROFILE</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>automatic</td>
<td>fallback</td>
<td>No</td>
<td>compiler_default</td>
<td>0</td>
<td>No</td>
<td>No</td>
<td>ex-mm</td>
</tr>
<tr>
<td>ex-trace</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>automatic</td>
<td>fallback</td>
<td>No</td>
<td>compiler_default</td>
<td>0</td>
<td>No</td>
<td>No</td>
<td>ex-mm</td>
</tr>
<tr>
<td>ex-sample</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>never</td>
<td>never</td>
<td>No</td>
<td>compiler_default</td>
<td>0</td>
<td>No</td>
<td>No</td>
<td>ex-mm</td>
</tr>
</tbody>
</table>

### Projects (/home/jlinford/.tau)

<table>
<thead>
<tr>
<th>Name</th>
<th>Targets</th>
<th>Applications</th>
<th>Measurements</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex-mm</td>
<td>localhost</td>
<td>ex-mm-serial</td>
<td>ex-profiler</td>
<td>/home/jlinford/.tau</td>
</tr>
<tr>
<td></td>
<td>ex-mm-openmp</td>
<td>ex-trace</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ex-mm-openmp-mpi</td>
<td>ex-sample</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### No trials. Use `tau <command>` or `tau trial create <command>` to create a new trial

```bash
jlinford@east03 ~/workspace/tauclmr/examples/mm $ tau dash
```
First use on a “vanilla” system

Put \texttt{tau} in front of every command

Detects, downloads, and installs required dependencies

Configures environment, wraps compiler
Executions create experiment trials

Put `tau` in front of every command

`tau show` to see data from last trial
Executions create experiment trials

<table>
<thead>
<tr>
<th>Name</th>
<th>Host OS</th>
<th>Host Arch</th>
<th>C</th>
<th>C++</th>
<th>Fortran</th>
<th>In Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost</td>
<td>Linux</td>
<td>x86_64</td>
<td>/usr/bin/gcc</td>
<td>/usr/bin/g++</td>
<td>/usr/bin/gfortran</td>
<td>ex-mm</td>
</tr>
</tbody>
</table>

**Applications (/home/jlinford/.tau)**

<table>
<thead>
<tr>
<th>Name</th>
<th>OpenMP</th>
<th>Pthreads</th>
<th>MPI</th>
<th>CUDA</th>
<th>MIC</th>
<th>SHMEM</th>
<th>MPC</th>
<th>In Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex-mm-serial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ex-mm-openmp</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ex-mm-openmp-mpi</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Measurements (/home/jlinford/.tau)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Profile</th>
<th>Trace</th>
<th>Sample</th>
<th>Source Inst.</th>
<th>Compiler Inst.</th>
<th>MPI</th>
<th>OpenMP</th>
<th>Callpath Depth</th>
<th>Mem. Usage</th>
<th>Mem. Alloc</th>
<th>In Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex-profile</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>automatic</td>
<td>fallback</td>
<td>No</td>
<td>compiler_default</td>
<td>0</td>
<td>No</td>
<td>No</td>
<td>ex-mm</td>
</tr>
<tr>
<td>ex-trace</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>automatic</td>
<td>fallback</td>
<td>No</td>
<td>compiler_default</td>
<td>0</td>
<td>No</td>
<td>No</td>
<td>ex-mm</td>
</tr>
<tr>
<td>ex-sample</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>never</td>
<td>never</td>
<td>No</td>
<td>compiler_default</td>
<td>0</td>
<td>No</td>
<td>No</td>
<td>ex-mm</td>
</tr>
</tbody>
</table>

**Projects (/home/jlinford/.tau)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Targets</th>
<th>Applications</th>
<th>Measurements</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex-mm</td>
<td>localhost</td>
<td>ex-mm-serial</td>
<td>ex-profile</td>
<td>/home/jlinford/.tau</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ex-mm-openmp</td>
<td>ex-trace</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ex-mm-openmp-mpi</td>
<td>ex-sample</td>
<td></td>
</tr>
</tbody>
</table>

Each execution is a new trial

2 trials of 'mm' (22.6KiB). Use `tau trial list` to see details.

jlinford@east03 ~/workspace/taucmdr/examples/mm $
Changing from serial to MPI+OpenMP

== Projects (/home/jlinford/tau) ===============================================

<table>
<thead>
<tr>
<th>Name</th>
<th>Targets</th>
<th>Applications</th>
<th>Measurements</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex-mm</td>
<td>localhost</td>
<td>ex-mm-serial</td>
<td>ex-profile</td>
<td>/home/jlinford/tau</td>
</tr>
<tr>
<td></td>
<td>localhost-openmpi</td>
<td>ex-mm-openmp</td>
<td>ex-trace</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ex-mm-openmp-mpi</td>
<td>ex-sample</td>
<td></td>
</tr>
</tbody>
</table>

== ex-mm (localhost-openmpi, ex-mm-openmp-mpi, ex-profile) Trials =============

No trials. Use 'tau <command>' or 'tau trial create <command>' to create a new trial.

jlinford@east03 ~/workspace/taucmdr/examples/mm $ tau mpicc *.c -fopenmp -o mm
Installing TAU at '/home/jlinford/tau/TAU/' from 'http://tau.uoregon.edu/tau.tgz' with 'aix-6_1' and MPI compilers
Using TAU source archive at '/home/jlinford/tau/src/tau.tgz'
Reusing TAU source files found at '/home/jlinford/tau/src/tau-2.24.1'
Configuring TAU with '-iowrapper...'
Compiling and installing TAU...
TAU installation complete
tau_cc.sh matmult.c matmult_initialize.c -fopenmp -o mm

jlinford@east03 ~/workspace/taucmdr/examples/mm $
Workflow is unchanged

```
`tau show` to see data from last trial
```
CONCLUSION
Downloads

http://tau.uoregon.edu

http://github.com/ParaToolsInc/taucmdr

http://www.hpclinux.com

Free download, open source, BSD license
Acknowledgements

- Department of Energy
  - Office of Science
  - Argonne National Laboratory
  - Oak Ridge National Laboratory
  - NNSA/ASC Trilabs (SNL, LLNL, LANL)

- HPCMP DoD PETTT Program

- National Science Foundation
  - Glassbox, SI-2

- University of Tennessee

- University of New Hampshire
  - Jean Perez, Benjamin Chandran

- University of Oregon
  - Allen D. Malony, Sameer Shende
  - Kevin Huck, Wyatt Spear

- TU Dresden
  - Holger Brunst, Andreas Knupfer
  - Wolfgang Nagel

- Research Centre Jülich
  - Bernd Mohr
  - Felix Wolf
Intuitive Performance Engineering

REFERENCE
Online References

- PAPI:
  - PAPI documentation is available from the PAPI website:
    http://icl.cs.utk.edu/papi/

- TAU:
  - TAU Users Guide and papers available from the TAU website:
    http://tau.uoregon.edu/

- VAMPIR:
  - VAMPIR website:
    http://www.vampir.eu/

- Scalasca:
  - Scalasca documentation page:
    http://www.scalasca.org/

- Eclipse PTP:
  - Documentation available from the Eclipse PTP website:
    http://www.eclipse.org/ptp/
• If your Fortran code uses free format in .f files (fixed is default for .f):
  % export TAU_OPTIONS='-optPdtF95Opts="-R free" -optVerbose'

• To use the compiler based instrumentation instead of PDT (source-based):
  % export TAU_OPTIONS='-optCompInst -optVerbose'

• If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):
  % export TAU_OPTIONS='-optPreProcess -optVerbose'

• To use an instrumentation specification file:
  % export TAU_OPTIONS=
     '-optTauSelectFile=select.tau -optVerbose -optPreProcess'

**Example select.tau file**

```
BEGIN_INSTRUMENT_SECTION
loops file="*" routine="#"
memory file="foo.f90" routine="#"
io file="abc.f90" routine="FOO"
END_INSTRUMENT_SECTION
```
% export TAU_MAKEFILE=$TAU/Makefile.tau-bgqtimers-papi-mpi-pdt
% export TAU_OPTIONS='--optTauSelectFile=select.tau --optVerbose'
% cat select.tau

BEGIN_INSTRUMENT_SECTION
  loops routine="#"
END_INSTRUMENT_SECTION

% export PATH=$TAU_ROOT/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
%
% qsub --env TAU_METRICS=TIME:PAPI_FP_INS:PAPI_L1_DCM -n 4 -t 15 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Metrics Panel -> “PAPI_FP_INS”, click “/”, “TIME”, click “Apply” and choose the derived metric.
Tracking I/O in static binaries

% export TAU_MAKEFILE=$TAU/Makefile.tau-bqgtimers-papi-mpi-pdt
% export PATH=$TAU_ROOT/bin:$PATH
% export TAU_OPTIONS=‘-optTrackIO -optVerbose’
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
% mpirun -n 4 ./a.out
% paraprof -pack ioprofile.ppk
% export TAU_TRACK_IO_PARAMS 1
% mpirun -n 4 ./a.out (to track parameters used in POSIX I/O calls as context events)
Installing and Configuring TAU

• Installing PDT:
  – wget http://tau.uoregon.edu/pdt.tgz
  – .configure --prefix=<dir>; make ; make install

• Installing TAU:
  – wget http://tau.uoregon.edu/tau.tgz
  – ./configure --bfd=download --pdt=<dir> --papi=<dir> ...
  – make install

• Using TAU:
  – export TAU_MAKEFILE=<taudir>/<arch>/lib/Makefile.tau-
  --TAGS
  – make CC=tau_cc.sh  CXX=tau_cxx.sh  F90=tau_f90.sh
% tau_compiler.sh

- `optVerbose` Turn on verbose debugging messages
- `optCompInst` Use compiler based instrumentation
- `optNoCompInst` Do not revert to compiler instrumentation if source instrumentation fails.
- `optTrackIO` Wrap POSIX I/O call and calculates vol/bw of I/O operations
- `optMemDbg` Runtime bounds checking (see TAU_MEMDBG_* env vars)
- `optKeepFiles` Does not remove intermediate .pdb and .inst.* files
- `optPreProcess` Preprocess sources (OpenMP, Fortran) before instrumentation
- `optTauSelectFile=\"<file>\"` Specify selective instrumentation file for `tau_instrumentor`
- `optTauWrapFile=\"<file>\"` Specify path to `link_options.tau` generated by `tau_gen_wrapper`
- `optHeaderInst` Enable Instrumentation of headers
- `optTrackUPCR` Track UPC runtime layer routines (used with tau_upc.sh)
- `optPdtF95Opts=''` Add options for Fortran parser in PDT (f95parse/gfparse) ...
## Runtime Environment Variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU_TRACE</td>
<td>0</td>
<td>Setting to 1 turns on tracing</td>
</tr>
<tr>
<td>TAU_CALLPATH</td>
<td>0</td>
<td>Setting to 1 turns on callpath profiling</td>
</tr>
<tr>
<td>TAU_TRACK_MEMORYLeaks</td>
<td>0</td>
<td>Setting to 1 turns on leak detection (for use with –optMemDbg or tau_exec)</td>
</tr>
<tr>
<td>TAU_MEMDBG_PROTECT_ABOVE</td>
<td>0</td>
<td>Setting to 1 turns on bounds checking for dynamically allocated arrays. (Use with –optMemDbg or tau_exec –memory_debug)</td>
</tr>
<tr>
<td>TAU_CALLPATH_DEPTH</td>
<td>2</td>
<td>Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)</td>
</tr>
<tr>
<td>TAU_TRACK_IO_PARAMS</td>
<td>0</td>
<td>Setting to 1 with –optTrackIO or tau_exec –io captures arguments of I/O calls</td>
</tr>
<tr>
<td>TAU_TRACK_SIGNALS</td>
<td>0</td>
<td>Setting to 1 generate debugging callstack info when a program crashes</td>
</tr>
<tr>
<td>TAU_COMM_MATRIX</td>
<td>0</td>
<td>Setting to 1 generates communication matrix display using context events</td>
</tr>
<tr>
<td>TAU_THROTTLE</td>
<td>1</td>
<td>Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently</td>
</tr>
<tr>
<td>TAU_THROTTLE_NUMCALLS</td>
<td>100000</td>
<td>Specifies the number of calls before testing for throttling</td>
</tr>
<tr>
<td>TAU_THROTTLE_PERCALL</td>
<td>10</td>
<td>Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call</td>
</tr>
<tr>
<td>TAU_COMPENSATE</td>
<td>0</td>
<td>Setting to 1 enables runtime compensation of instrumentation overhead</td>
</tr>
<tr>
<td>TAU_PROFILE_FORMAT</td>
<td>Profile</td>
<td>Setting to “merged” generates a single file. “snapshot” generates xml format</td>
</tr>
<tr>
<td>TAU_METRICS</td>
<td>TIME</td>
<td>Setting to a comma separated list generates other metrics. (e.g., TIME:P_VIRTUAL_TIME:PAPI_FP_INS:PAPI_NATIVE_&lt;event&gt;:&lt;subevent&gt;)</td>
</tr>
</tbody>
</table>